Egyptian Informatics Journal (2016) 17, 161–173

Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

ORIGINAL ARTICLE

Design and analysis of stochastic DSS query optimizers in a distributed database system

Manik Sharma [a,*], Gurvinder Singh [b], Rajinder Singh [b]

[a] Department of Computer Science and Applications, DAV University, Jalandhar, India
[b] Department of Computer Science, Guru Nanak Dev University, Amritsar, India

Received 7 June 2015; revised 17 August 2015; accepted 26 October 2015
Available online 19 November 2015

KEYWORDS

Query optimization;
Total Costs;
DSS;
Stochastic approach, entropy;
Genetic Algorithms

Abstract Query optimization is a stimulating task of any database system. A number of heuristics have been applied in recent times, which proposed new algorithms for substantially improving the performance of a query. The hunt for a better solution still continues. The imperishable developments in the field of Decision Support System (DSS) databases are presenting data at an exceptional rate. The massive volume of DSS data is consequential only when it is able to access and analyze by distinctive researchers. Here, an innovative stochastic framework of DSS query optimizer is proposed to further optimize the design of existing query optimization genetic approaches. The results of *Entropy Based Restricted Stochastic Query Optimizer* (*ERSQO*) are compared with the results of *Exhaustive Enumeration Query Optimizer* (*EAQO*), *Simple Genetic Query Optimizer* (*SGQO*), *Novel Genetic Query Optimizer* (*NGQO*) and *Restricted Stochastic Query Optimizer* (*RSQO*). In terms of Total Costs, *EAQO* outperforms *SGQO, NGQO, RSQO* and *ERSQO*. However, stochastic approaches dominate in terms of runtime. The Total Costs produced by *ERSQO is better than SGQO, NGQO* and *RGQO* by 12%, 8% and 5% respectively. Moreover, the effect of replicating data on the Total Costs of DSS query is also examined. In addition, the statistical analysis revealed a 2-tailed significant correlation between the number of join operations and the Total Costs of distributed DSS query. Finally, in regard to the consistency of stochastic query optimizers, the results of *SGQO, NGQO, RSQO and ERSQO* are 96.2%, 97.2%, 97.45 and 97.8% consistent respectively.

1. Introduction

Database is a systematic collection of closely interrelated data designed to converge the information demands of a business organization. It is a repository of the data that can be shared and accessed by concurrent users. It holds the operational data as well as its comprehensive description. Last few decades witnessed the phenomenal progress in the database technology. There has been an outstanding change in the way many

* Corresponding author.
E-mail address: manik_sharma25@yahoo.com (M. Sharma).
Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2015.10.003

organizations operate and manage data. There is a substantial progress in the number of database users and the organizations that use it. Database system is a comprehensive collection of interrelated programs used to successfully create, store, retrieve and manage data in the database. Earlier, the complete database was supposed to be placed on a server machine, and was shared among database users. This approach of database management was well recognized as centralized database management. This approach had considerably overwhelmed the various problems (Data Redundancy, No Sharing, Security, Inconsistency, etc.) of traditional file based systems. Later, it was seriously contemplated that storing the entire database on a single site was one of the major bottlenecks of this system, as it degraded the performance of the system. Moreover, this approach failed to provide faster 'Access Time' and 'Response Time' as the size of database was increased to the significant higher levels. In 1980s, with the fusion of database system and computer networking, a new term called distributed database system emerged. It was one of the major developments of database technology. Several problems of traditional database systems were resolved by the distributed database system.

Query is a statement or group of statement that adequately performs some basic database operations viz. 'Read', 'Write', 'Delete', and 'Update'. It plays a consequential role in managing and retrieving data. In general, distributed queries are more complex and complicated as compared to centralized queries. A distributed query is further classified as Online Transaction Processing (OLTP), and Decision Support System (DSS) queries. *DSS* query is distributed in nature. In general, it is complicated and takes more execution time. These queries can access data from a local as well as from remote sites. These queries normally deal with substantial volume of data as compared to *OLTP* queries. *DSS* queries use up ample quantity of input–output, processing and communication resources and can abruptly halt CPU or even memory server of a distributed database system. Moreover, the running time of distributed *DSS* query is conventionally unforeseeable. *DSS* queries work on relations having a size in mega bytes, giga bytes or even larger in size. Query optimization finally revolved out to be the biggest challenge for the database researchers. Query optimization in distributed database system has gained considerable attention in recent years. It is a process to determine the best query execution plan in terms of *Total Costs* or *Response Time*. A non-optimal query execution plan is costly either in terms of usage of system resources or in terms of execution time. In this research work, the focus is to analyze the performance of different stochastic distributed DSS query optimizers. The results are compared based upon usage of system resources and runtime of the distributed DSS query. Here, two novel stochastic query optimizers viz. Restricted Stochastic Query Optimizer (RSQO) and Entropy based Restricted Stochastic Query Optimizer (ERSQO) have been proposed to optimize a set of distributed DSS queries. The results of ERSQO and RSQO are compared with SGQO and NGQO.

The paper is divided into several sections. Section 2 of the paper highlighted the related work. Research problem is formulated in Section 3. The basic concept of query optimization is depicted in Section 4. Section 5 represents the design of different stochastic query optimizers. The design of cost coefficients and experimental setup has been described in Section 6. Section 7 represents discussion and results. Conclusion has been made in Section 8. Finally, future research scope is mentioned in Section 9.

2. Related works

The credit of query optimization goes to Yao and Hevner. In the late 1970s, authors used heuristic with exhaustive enumeration approach to optimize the queries. In 1980s different key researcher's viz. Ceri and Palagatti, Chen and Li, Yu and Chang, Peter Apers, Lam and Martin proposed the different query optimization strategies. *Rho and March* further extend the query optimization model in 1995. In the 21st century, the researcher's like *Ahmat Cosar, Zehai Zhou* used 'Genetic Algorithm' to optimize the distributed queries [1–5,29]. Traditionally 'Exhaustive Enumeration' with some heuristics algorithms (Dynamic Programming, Branch and Bound, Greedy Algorithm, etc.) was dominantly used to optimize queries. However, this technique was not fit for large and complex queries as it almost failed to provide an optimal query allocation plan in a finite amount of time. It took minutes, hours, or even days to provide an optimal query execution plan for a complex query [6,7]. In randomized optimization strategies, an optimal solution was found by a set of random moves. Every solution of search space was represented as a solution point. Random moves were used to connect one solution point to another as an edge. The set of random moves heavily depends upon the nature of optimization problem and set of solutions. 'Iterative Improvements', 'Simulated Annealing', 'Random Sampling', etc. are some of the examples of randomized optimization strategies. In recent times, evolutionary techniques are used in optimizing the distributed queries. Evolutionary techniques are based on the evolution of a population. Some of the important features of evolutionary techniques are tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness, low solution cost and better rapport with reality. Some of the commonly used evolutionary techniques are 'Genetic Algorithms', 'Swarm Intelligence', 'Memetic Algorithms', 'ACO', etc [8–11].

3. Problem formulation

Query optimization is an NP-Hard problem. A number of heuristics have been applied in recent times, which propose new techniques to optimize the query processing cycle. The hunt for better solutions still continues. In general, a query can be optimized by changing the execution order of subqueries, restructuring a query in different ways or with the effective allocation of sub-operations to distinctive sites (*operation site allocation*). Operation site allocation problem is one of the prominent distributed database research problems that continue to spur a massive deal of attention. Therefore, in this research work, while optimizing a query, the primary focus has been given to allocate sub operation to different sites of a distributed database network. Decision support system queries process massive amount of data (in gigabytes, petabytes or even more). These queries are not subjected to the response time. However, usage of system resources required to execute the query is of major concern. Therefore, *DSS* queries are optimized on the basis of *Total Costs* (*Sum of input output, processing and communication Costs*). The problem is devised as below:

If
B represents a set of relations
$B = \{b_1, b_2, b_3, \ldots, b_n\}$
S is set of network sites where base relations are placed
$S = \{s_1, s_2, \ldots, s_n\}$
Q is a set of decision support system queries based upon TPC-DS benchmark
$Q = \{q_1, q_2, q_3, \ldots, q_n\}$
Then
An objective function

$$T\_Costs_{DSS}(Total\ Costs\ of\ DSS\ Query) = T\_Costs_{io} + T\_Costs_{cpu} + T\_Cost_{comm} \quad (1.1)$$

Minimize (T_Costs_{DSS})
where,
T_Costs_{io} is the total input output costs i.e. total time required for I/O operations.
T_Costs_{cpu} is the total processing costs i.e. total time required for processing.
$T\_Costs_{comm.}$ is total communication costs i.e. total time required for data transfer.

A distributed DSS query optimizer has been designed to solve the operation site allocation problem of distributed DSS queries. For finding an optimal operation site allocation plan, first of all, a 'SQL' based decision support system query is decomposed into relational algebra expressions (sub-operations) based on 'selection', 'projection', 'join' and 'semi-join'. These sub-operations are then allocated to different sites for their execution by exploring various amalgams of operations and sites. The costs of each sub-operation are computed by using the size of relation/fragment involved in the query, site allocated and the values of costs coefficients of input–output, processing and communication. The operation site allocation problem is represented in Fig. 1.

Here, a DSS query has been optimized using exhaustive enumeration, stochastic, restricted stochastic and entropy based restricted stochastic approaches. Moreover, the effect of data replication on the distributed query optimization process has examined. An effort is also made to analyze the correlation between number of joins operation and the 'Total Costs' of the distributed DSS query. Finally, the consistency of DSS query optimizer has been statistically analyzed.

4. DSS query optimization

Query optimization is a process that generates the different operation site allocation plans to execute the query. The objective of operation site allocation problem is to select a better query execution plan which optimizes the *Total Costs* of the distributed decision support system query. In commercial database systems, the query optimization routine is mechanized by a software module called query optimizer. A query optimizer is composed of three components viz. 'Cost Model', 'Search Space' and 'Search Strategy' [2,12]. The search space represents a set of alternative query execution plans for a query. To find an optimal or best possible operation site allocation plan, the different execution plans generated by query optimization approach are compared on the basis of the 'Total Costs' of a query. The cost model is responsible for associating the 'Costs' with each query execution plan. The costs are generated on the basis of the operation and execution environment of a query. In common practice, 'Costs' are represented by a 'Costs Function' also called objective function. It is normally constructed on the basis of the usage of system resources or the execution time of a query. It is significantly affected by different factors such as speed of input output devices, network media, size and cardinality of relations, and size of blocks. The role of search strategy is to find the best possible query execution plan by probing the search space [13].

5. Design of DSS query optimizers using stochastic approaches

On the basis of earlier works, it was found that the scaled up NP-hard problems are almost intractable to solve using exhaustive enumeration techniques. However, it can be effectively solved by stochastic approaches. *Genetic Algorithm* (*GA*) is a prevalent stochastic approach. In *GA*, the time taken in obtaining solution is independent of the search space. Therefore, genetic approach is best suited for optimizing the queries in a distributed environment [8,14,15,10,16].

'Genetic Algorithm' commonly abbreviated as 'GA' was proposed by John Holland. These are search algorithms specifically designed to simulate the principle of the natural biological evolution process. 'GA' borrows its essential features from

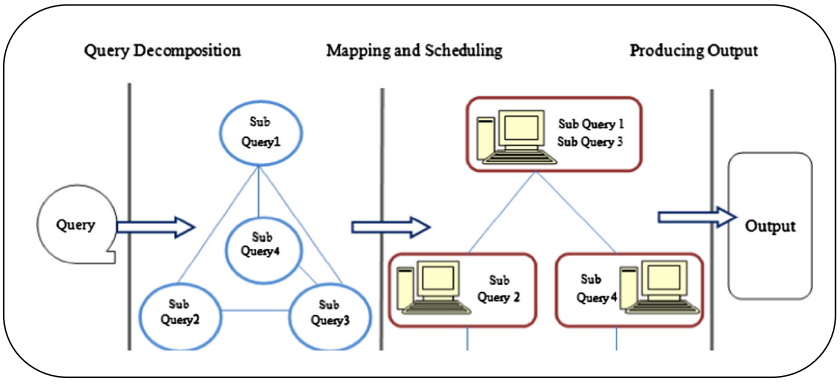

Figure 1 Query processing in distributed database system.

natural genetics. In other words, genetic algorithms are stochastic techniques that stipulate good-quality solution with low time complexity. It permits a population composed of many individual chromosomes to evolve under delineated selection rules to generate a state that optimizes the objective function. These types of algorithms successfully operate on a population of solutions rather than a single solution. It generally employs some heuristics such as selection, crossover and mutation to develop better solutions [16,8].

'Genetic Algorithms' are capable of being applied to an enormously wide range of problems. Some of the major applications of these algorithms are Image Processing, Environmental Sciences, Analysis of Time Series, Task Scheduling, Bioinformatics, Clustering, Game Theory, Artificial Intelligence, Aeronautics, etc. [17–22]. In general, these types of algorithms aim for searching better solution from a number of available solutions. As stated, GA starts its working from a set of solutions rather than a single solution. The initial population is generated randomly. Each solution of the problem is adequately represented by encoding a string (Chromosome) of bits or characters (Genes). Every chromosome has a fitness value associated with it. The collection of chromosomes with their corresponding fitness values is called population. The population at a particular instance is called generation. Fitness function is one of the major decisive parameters of 'Genetic Algorithm'. It defines the objective of the problem to be optimized. A pair of chromosomes based on their fitness values is used to reproduce offsprings. The genetic properties of both the chromosomes are intermixed to generate better offspring, such a mechanism is called crossover. After crossover operation, the genetic characteristics of the generated offspring are further modified. Mutation is a procedure to modify the properties of the generated offspring to make it more effective. The algorithm terminates when the required condition is fulfilled [16,23].

A simple pseudo-code for a genetic algorithm is given below:

```
Gen = 0;                          // Generation Counter
Initiate_Poulation POP (Gen);     // Initial population generation
Fitness_EvaluationPOP (Gen);      // Evaluates Individual's Fitness
While Gen < Max_Gen               // Terminating Criterion
Begin
Gen = Gen + 1                     // Increase Generation Counter
  Select Parents                  // Select Parents from the population
  Crossover;                      // Apply crossover to selected parents
  Mutation                        // Apply Mutation to offsprings
  Evaluate_Fitness;               // Evaluates Fitness of offspring
  Select best Offspring for next  // Select Survivor Individuals
Generation
End
```

In this research paper, an effort has been made to design and implement four different variations of Genetic Algorithms. Initially, the design of Rho and March (SGQO) and Sevinc and Cosar (NGQO) is implemented as SGQO and NGQO respectively. Furthermore, two restricted stochastic query optimizers have been proposed as RSQO and ERSQO. ERSQO uses the concept of Genetic Algorithm and Havrda and Charvat entropy. The remaining section explains the design and working of different stochastic distributed DSS query optimizers.

5.1. Design of Exhaustive Enumeration Query Optimizer (EAQO)

Exhaustive Enumeration approach is a deterministic technique that accomplishes a complete search of solution space. It generates and inspects all the possible combinations of search space that is assured to provide the best promising solution. It is quite easy to understand and implement it. However, it is inelegant to solve large and complex problems. While solving a DSS Operation Site Allocation problem, it explores all possible combinations of query execution plans. The design of *Exhaustive Enumeration Query Optimizer* (*EAQO*) for solving the above said research problem in distributed database environment is based upon the following decision variables:

NoS:	Number of sites
NoB:	Number of base relations
NoO:	Total number of operations involved in a query
NoF:	Number of intermediate fragments
NoSo:	Number of selection operations
NoPo:	Number of projection operations
NoJo:	Number of join operations
T_Costs_{io}:	Total input–output costs
T_Costs_{cpu}:	Total processing costs
T_Costs_{comm}:	Total communication costs
Res_Site:	Resultant site for final operation

Psuedo-Code for solving operation site allocation problem using Simple Exhaustive Enumeration

```
// Input Data
Read a DSS Query
Read various decision variables viz. NoS, NoB, NoO, NoJo, NoSo, NoPo, NoF
Read Res_Site
// Generation of Query plans
N = NoO − 1
  For K = 1 to N
    For A[K] = 1 to NoS
      For A[K + 1] = 1 to NoS
        For A[K + 2] = 1 to NoS
          ...
          ...
        For A[N] = 1 to NoS
        Compute T_Costs_io for selection, Projection & Join
        Compute T_Costs_cpu (Processing Costs) for Selection, Projection & Join
        Compute T_Costs_comm (Communication Costs) for Join
        T_Costs_DSS = T_Costs_io + T_Costs_cpu + T_Costs_comm.
        End
        End
  ...
  End
  End
```

5.2. Design of Simple Genetic Query Optimizer (SGQO)

On the basis of Rho and March [14] approach, *Simple Genetic Query Optimizer (SGQO)* has been designed for solving the operation site allocation problem of distributed DSS queries. *SGQO* starts with randomly generated initial population. A chromosome is designed on the basis of number of operations and number of sites. The chromosome has been designed in way that its length is one less than the number of operations of a query [14]. The pseudo-code for *SGQO* is formulated as below:

// Input Data
Read a TPC-DS based Adhoc DSS query.
Decompose a query into distinct sub operations viz. Selection, Projection and Join.
Read various Input variables viz. NoS (Number of Sites), NoB (Number of Base Relations), NoO (Number of operations), NoJ (Number of join operations), NoF (Number of intermediate fragments), NoSo (Number of selection Operations), NoPo (Number of projection operations), IoC (Input output costs coefficients), CP (Processing costs coefficients), Comm (Communication costs coefficients), PopSize (PopSize),MaxGenr (Number of Generations).

// Initial Population
Design chromosome having length one less than the number of operations.
Randomly generate an initial Population by using the concept of roulette wheel selection with PopSize number of chromosomes.

// Selection Operation
Select any two chromosomes that act as parent to perform crossover and mutation operations.

// Crossover Operation
Apply one-point crossover operation over two selected chromosomes.

// Mutation Operation
Apply mutation operation on the resultant of crossover operation and store it as a member of new generation.

// Analyze the fitness
$T\_Costs_{DSS} = T\_Costs_{io} + T\_Costs_{cpu} + T\_Costs_{comm}$
Compute the fitness value of the chromosome based upon T_Costs_{DSS}.

// Termination
Generate DSS query allocation plan and go to step (Analyze the Fitness) until MaxGenr.

5.3. Design of Novel Genetic Query Optimizer (NGQO)

Sevinc and Cosar [8] proposed a *Novel Genetic Query Optimizer* (*NGQO*) for optimizing the distributed queries in a novel way. *NGQO* improved the quality of solution in finding an optimal query execution plan by forbidding the redundant chromosome while performing crossover and mutation operations. On the basis of Sevinc and Cosar approach, *NGQO* is designed for optimizing distributed *DSS* queries [8]. The pseudo-code of NGQO is given below:

// Input Data
Read a TPC-DS based Adhoc DSS query.
Decompose a query into sub queries based upon different operations like selection, projection and join.
Read various Input variables viz. NoS (Number of Sites), NoB (Number of Base Relations), NoO (Number of operations), NoJ (Number of join operations), NoF (Number of intermediate fragments), NoSo (Number of selection Operations), NoPo (Number of projection operations), IoC (Input output costs coefficients), CP (Processing costs coefficients), Comm (Communication costs coefficients), PopSize (PopSize),MaxGenr

// Initial Population
Design chromosome having length one less than the number of operations.
Randomly generate an initial population by using the concept of roulette wheel selection with PopSize number of chromosomes.

// Selection Operation
Select any two chromosomes which were not used earlier to perform crossover and mutation operations.

// Crossover Operation
Apply one-point crossover operation over two selected parents.

// Mutation Operation
Apply mutation operation on the resultant of crossover operation and store it as a member of new generation.

// Analyze the fitness
$T\_Costs_{DSS} = T\_Costs_{io} + T\_Costs_{cpu} + T\_Costs_{comm}$
Compute the fitness value of the chromosome based upon T_Costs_{DSS}.

// Termination
Generate DSS query allocation plan and go to step (Analyze the Fitness) until MaxGenr.

5.4. Design of Restricted Stochastic Query Optimizer (RSQO)

Like *SGQO and NGQO, RSQO* also starts with randomly generated initial population. A chromosome is designed to allocate sub-operations of a *DSS* query on a distributed network. The innovation of approach lies in the restricted growth of the chromosome design. Here, projection sub-operations is supposed to be executed on the same machine where the corresponding selection operations were executed. This design of chromosome reduced the 'Processing Costs' of the query, which further reduced the *Total Costs* of the DSS query. The three basic operators of 'GA' viz. 'Selection', 'Crossover' and 'Mutation' are modified. The quality of solution produced by *RSQO* is better than as given by *SGQO and NGQO*. However, like other stochastic approaches, it does not guarantee the best solution as compared to *Simple Exhaustive Enumeration* and *Restricted Exhaustive Enumeration* approach [24].

The pseudo-code of *RSQO* is as given below:

*// **Input Data***
Read a TPC-DS based Adhoc DSS query.
Decompose a query into sub queries based upon different operations like Selection, Projection and Join
Read various Input variables viz. NoS (Number of Sites), NoB (Number of Base Relations), NoO (Number of operations), NoJ (Number of join operations), NoF (Number of intermediate fragments), NoSo (Number of selection Operations), NoPo (Number of projection operations), IoC (Input output costs coefficients), CP (Processing costs coefficients), Comm (Communication costs coefficients), PopSize (PopSize),MaxGenr

*// **Initial Population***
Design a chromosome in such a way that the projection operation of a distributed DSS query is executed on the same site where corresponding selection operation was carried out.
The length of chromosome must be one less than the total number of operations.
Randomly generate an initial population with above designed chromosome by using the concept of roulette wheel selection.

*// **Selection Operation***
Select any two chromosomes having uniform probability that act as parent to perform crossover and mutation operations.

*// **Crossover Operation***
Apply one-point crossover operation on the Join operation portion of chromosome over two selected parents.

*// **Mutation Operation***
Apply mutation operation on the resultant of crossover operation in such a way that it does not affect restriction imposed over the execution of projection operation.
Store resultant as a member of new generation.

*// **Analyze the fitness***
$T\_Costs_{DSS} = T\_Costs_{io} + T\_Costs_{cpu} + T\_Costs_{comm}$
Compute the fitness value of the chromosome based upon T_Costs_{DSS}.

*// **Termination***
Generate DSS query allocation plan and go to step (Analyze the Fitness) until MaxGenr.

5.5. Design of Entropy based Restricted Stochastic Query Optimizer (ERSQO)

To improve the design of *RSQO*, an *Entropy based Restricted Stochastic Query Optimizer* is proposed. In ERSQO, the innovation lies in the restricted growth of chromosome and the use of *Havrda and Charvat* entropy. Here, entropy is used at two different levels. Firstly, the concept of entropy is implemented at the selection operator of *ERSQO* so that every member of *Population/Generation* has uniform probability of selecting as a parent to perform crossover and mutation operations. The concept of entropy is also used while selecting a site for executing the sub-operations of a *DSS* query. Here each permissible site has uniform probability of its selection. Furthermore *Havrda and Charvat entropy* is used to refrain low diversity population problem which normally occurs in the implementation of 'Genetic Algorithm'. The low diversity population problem deteriorates the quality of the stochastic approach. The diversity of all chromosomes is measured by using the following formula [25–27,31].

$$H(P) = \frac{1}{1-\alpha}\sum_{k=1}^{n} P^n - 1 \tag{1.2}$$

In general, P can be represented as P_{ij}. Here

$$P_{ij} = nS_{ij}/PopSize.$$

nS_{ij} denotes the number of placements of site j on the locus of i.

$H(P)$ approaches to maximum values $Max = (PopSize^{1-\alpha} - 1)/(1-\alpha)$ when each site of a distributed database system involved in *DSS* query appears uniformly in the population. On the other hand, $H(P)$ tends to minimum or zero when all the sites involved in a *DSS* query lie on the same locus or path of all chromosomes [25–27]. The pseudo-code of *Entropy based Genetic Algorithm* with restricted growth encoding scheme is as given below:

*// **Input Data***
Read a DSS query, and break down it into sub operations like selection, projection and join.
Read various Input variables viz. NoS (Number of Sites), NoB (Number of Base Relations), NoO (Number of operations), NoJ (Number of join operations), NoF (Number of intermediate fragments), NoSo (Number of selection Operations), NoPo (Number of projection operations), IoC (Input output costs coefficients), CP (Processing costs coefficients), Comm (Communication costs coefficients), PopSize (PopSize),MaxGenr

*// **Initial Population***
Design chromosome having length one less than the number of operations.
Randomly generate an initial Population by using the concept of roulette wheel selection with PopSize number of chromosomes.

*// **Examine the Diversity of Population***
***Call** Havrda_Charvat_Entropy*

*// **Selection Operation with Entropy***
Select any two chromosomes having uniform probability that act as parent to perform crossover and mutation operations.

*// **Crossover Operation***
Apply One-point crossover operation over two selected parents.

*// **Mutation Operation***
Apply mutation operation on the resultant of crossover operation and store it as a member of new generation.

*// **Analyze the fitness***
$T\_Costs_{DSS} = T\_Costs_{io} + T\_Costs_{cpu} + T\_Costs_{comm}$
Compute the fitness value of the chromosome based upon T_Costs_{DSS}.

*// **Termination***
Generate DSS query allocation plan and Go to step (Analyze the Fitness) until MaxGenr.

//Procedure to Check the diversity of population using Havrda and Charvat Entropy

Procedure (Havrda_Charvat_Entropy)

I = 0
For J = 1 to N
Compute $H(P) = \frac{1}{1-\alpha}\sum_{k=1}^{n} P^n - 1$
if $H(P) < (PopSize^{1-\alpha} - 1)/(1-\alpha)$
I = I + 1
End if
End for
If I > n/cp
(Here *cp* is the control parameter, higher value of cp means better improvement)
Diversity of Population is low
Randomly generate new population based upon Input Parameters by using the Roulette Wheel Selection with average or high diversity
End if

5.6. Differences between the different query optimization approaches

Five different query optimization approaches to optimize distributed DSS queries in a distributed environment have been discussed. From the above subsections, it is found as exhaustive enumeration scans all different possible query execution plans, hence it is not feasible to use EAQO for complex distributed DSS queries. SGQO being stochastic is able to give an optimal query execution plan for moderate to complex distributed DSS query, however for complex moderate to complex DSS query may have redundant chromosomes. NGQO avoid redundant in chromosome, but still may lead to low diversity problem. RGQO is introduced to further improve the design of NGQO by using the restricted design of the chromosomes. On the other hand, ERSQO has been implemented by using the hybrid idea of restricted growth chromosome design and entropy to solve the low diversity population problem.

6. Experimental setup

For analyzing the efficiency and performance of different DSS query optimizers, following set of adhoc distributed DSS queries has been designed. The queries are focused on TPC-DS a benchmark database which is based upon customer and sales. The queries are represented in the form of relational algebra expressions. A set of queries is selected in such a way that in experimentation one has different number of join operations in each query. The queries are designed over a range of one-to-ten join operations. The queries are selected in such a way to vary the number of join operations. The queries are fired on a distributed database consisting of relations viz. Customer, Sales, Cust_Address, Marketing, Shipping, Webstore, Warehouse, Store, and Items [30].

DSS1: (π (σ) Customer): X: (π (σ) Cust_Address)
DSS2: (π (σ) Customer): X: (π (σ) Cust_Address): X: (π (σ) Sales)
DSS3: (π (σ)Customer): X: (π (σ)Sales): X: (π (σ)Warehouse): X: (π (σ) Marketing).
DSS4: (π (σ) Customer): X: (π (σ)Cust_Address): X: (π (σ) Sales): X: ((π (σ) Warehouse): X: (π (σ) Sales))
DSS5: (π (σ)Store): X: (π (σ) Customer): X: (π (σ) Cust_Address): X: (π (σ)Store): X: (π (σ)Sales): X: (π (σ) Items)
DSS6: (π (σ) Sales): X: (π (σ) Cust_Address): X: (π (σ) Sales): X: (π (σ)Item): X:(π (σ) Marketing): X: (π (σ)Sales): X: (π (σ)Shipping).
DSS7: (π (σ) Sales): X: (π (σ) Cust_Address): X: (π (σ) Items): X: (π (σ) Warehouse): X: (π (σ)Sales): X: (π (σ)Marketing): X: (π (σ)Shipping): X: (π (σ)Webstore)
DSS8: (π (σ) Sales): X: (π (σ)Cust_Address): X: (π (σ) Items): X: (π (σ) Warehouse): X: (π (σ)Sales): X: (π (σ)Marketing): X: (π (σ)Shipping): X: ((σ)Webstore: X: (π (σ) Items)
DSS9: (π (σ) Sales): X: (π (σ)Cust_Address): X: (π (σ) Items): X: (π (σ) Warehouse): X: (π (σ)Sales): X: (π (σ)Marketing): X: (π (σ)Shipping): X: (π (σ) Webstore: X: (π (σ) Items): X: (π (σ) call center)
DSS10: (π (σ) Sales): X: (π (σ) Items): X: (π (σ)Cust_Address): X: (π (σ) Customer): X: (π (σ) Store): X: (π (σ)Sales): X: (π (σ) Warehouse): X: (π (σ)Sales): X: ((σ) Marketing): X: (π (σ) Sales: X: (π (σ) shipping)

To solve operation site allocation problem, a simulator was developed to unravel the operation site allocation problem for a set of distributed *DSS* queries. It was designed using 'MATLAB 2008' environment without using the inbuilt 'GA' facilities. The system takes parameters of DSS query as an input and generates different query execution plans as an output. The system uses a number of input parameters such as number of base relations, total number of operations, number of selection operations, number of projection operations, number and size of intermediate fragments, costs coefficients of I/O, communication and processing, and number of join operations during the optimization process of a *DSS* query. The best possible query allocation plan that reduces the combined usage of I/O, CPU and communication resources requisite to execute a *DSS* query is selected as a final output. All the experiments were carried out based on the following assumptions [13,14].

- The computations were made based on the number of data blocks required by a query.
- Block size of a relation was assumed to be of 8 Kbytes.
- The base relation was replicated randomly on any two different sites. Size of intermediate fragments was calculated based on the selectivity estimation techniques.
- The default ratio of cost coefficients of input output and communication was assumed to be 1:1.6.
- 'Selection' and 'Projection' operations were processed on the sites where the corresponding base relation was placed.
- 'Join' operations were allowed to be executed on any site of a distributed database network.

Here, a set of distributed queries was designed assuming adhoc DSS queries. All the queries are focused on retrieval operations of a distributed database system. The queries are formulated by using selection, projection and join operations of relational algebra. 'Join' operation plays a prominent part in distributed database queries. Therefore, a set of queries is designed over a range of join operations. A series of experiments was conducted for a set of distributed *DSS* queries. The statistics of the set of distributed DSS queries is represented in Table 1.

As stated earlier, distributed database composed of distinctive base relations 'B' was simulated over a network of sites 'S'. 'Q' denotes a set of *DSS* queries to be optimized. 'q' is a *DSS* query designed to be analyzed by decomposing it into several sub-operations 'y'.

The costs coefficients of input output, processing and communication are designed on the basis of the 'Costs Model' of Rho and March, Dougless and Cornell, Sevinc and Cosar [8,14,28]. The design of 'Input–Output' costs coefficients is exhibited in the form of a linear array. The size of array is restricted by the number of sites available in a distributed database system. As per specification based on works of Rho and March, Sevinc and Cosar and Ozsu and Valduries the ratio of input–output costs coefficients to the communication costs coefficients has been taken as 1:1.6. Communication costs coefficients are depicted in the form of a square matrix of size

Table 1 Statistics of distributed DSS queries.

S. no.	Total Number of operation in a DSS query	Number of selection and projection operations	Number of join operations	Number of intermediate fragments	Number of relations	Number of sites
1	5	2, 2	1	6	2	2
2	8	3, 3	2	10	3	3
3	11	4, 4	3	14	4	4
4	14	5, 5	4	18	5	4
5	17	6, 6	5	22	6	4
6	21	7, 7	6	28	7	6
7	23	8, 8	7	30	8	10
8	27	9, 9	8	35	9	10
9	30	10, 10	9	39	10	10
10	34	11, 11	10	44	11	10

restricted by the number of sites. On the other hand, the ratio of processing costs coefficients to input output costs coefficients is 1:10. Like input–output costs coefficients, processing costs coefficients are also represented in the form of a linear array [13]. The prototype of different costs coefficients for a *DSS* query in a distributed database system consisting of ten sites is given below:

Input–output costs coefficients (I_{CC}):

10	11	12	13	14	15	12	10	11	10

Processing costs coefficients (P_{CC}):

1	1.1	1.2	1.3	1.4	1.5	1.2	1	1.1	1

Communication costs coefficients (CM_{CC}):

0	18	19	21	22	24	19	16	18	16
18	0	19	21	22	24	19	16	18	16
19	19	0	21	22	24	19	16	18	16
21	21	21	0	22	24	19	16	18	16
22	22	22	22	0	24	19	16	18	16
24	24	24	24	24	0	19	16	18	16
19	19	19	19	19	19	0	16	18	16
16	16	16	16	16	16	16	0	18	16
18	18	18	18	18	18	18	18	0	16
16	16	16	16	16	16	16	16	16	0

The costs coefficients for data allocation variables are represented in the form of a rectangular matrix having size of order of number of sites and base relations. Each relation is supposed to be replicated on two adjacent sites. The prototype of data allocation variable for ten sites and seven base relations is as given below:

Data allocation variables (DA_{bs}):

1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	1	1	0	0

Using the above designed decision variables and costs coefficients, one can find local processing costs (*LP_Costs*) and communication costs (*CM_Costs*) of a *DSS* query. Local processing costs is abbreviated as *LP_Costs* and *communication costs* is abbreviated as *CM_Costs*. Let T_Costs_{io} is the total input output costs and T_Costs_{cpu} is the total processing costs of a query.

Local processing costs are the sum of total input output costs (T_Costs_{io}) and total processing costs (T_Costs_{cpu}) of selection, projection and join operations of query. The input–output costs of the selection operation was computed by multiplying the input–output costs coefficients (I_{CC}) of site with the number of memory blocks accessed by a base relation 'b' i.e. size of intermediate fragments. Similarly, the 'Processing Costs' is generated by multiplying the processing costs coefficients (P_{CC}) of the site with the number of memory blocks read or written by a base relation '*b*'. Finally, the summation was used to find the total input output costs and total processing costs.

The communication costs is associated with join operations. The communication Costs of a query is computed as follows:

- In the first step, communication costs from the corresponding site of left child of 'Join' operation to the site where 'Join' operation is executed is selected. The communication costs then multiplied with the number of data blocks as given by 'Left Fragment' of the join operation.
- Similar calculation has been carried out for the right child of the 'Join' operation.
- Finally, the results of these two operations are added iteratively for the number of join operations involved in the query.

The mathematical formulation of local processing costs and communication costs in the proposed cost model is given below:

$$LP_Costs = \sum_{i=1}^{NoSo} I_{cc} * F_i + \sum_{j=1}^{NoPo} I_{cc} * F_j + \sum_{i=1}^{NoSo} P_{cc} * F_i + \sum_{j=1}^{NoPo} P_{cc} * F_j \quad (1.3)$$

The mathematical representation of the communication costs (*CMCT*) is as given below:

$$\text{CM\_Costs} = \sum_{i=1}^{Nojo} CM_{cc}(LPO, JO) * LPF_i + \sum_{i=1}^{Nojo} CM_{cc}(LPO, JO) * RPF_i \quad (1.4)$$

Here

JO	specifies the location of *Join* operation.
LPO and RPO	Left previous operation and right previous operation.
LPF & RPF	represents the left previous fragments and right previous fragment.
NoSo, NoPo and NoJo	represent the number of selection, projection and join operations of a query.

$$\text{T\_Costs}_{\text{DSS}} = \sum_{i=1}^{NoSo} I_{cc} * F_i + \sum_{j=1}^{NoPo} I_{cc} * F_j + \sum_{i=1}^{NoSo} P_{cc} * F_i + \sum_{j=1}^{NoPo} P_{cc} * F_j + \sum_{i=1}^{Noj} CM_{cc}(LPO, JO) * LPF_i + \sum_{i=1}^{Noj} CM_{cc}(LPO, JO) * RPF_i \quad (1.5)$$

The queries are optimized using four different stochastic approaches viz. 'Simple Genetic Approach', 'Novel Genetic Approach', 'Restricted Genetic Approach' and 'Entropy based Restricted Genetic Approach'. The optimization is based upon the Total Costs of the query. The outputs of the stochastic DSS query optimizers are compared with the results of EAQO. While optimizing the distributed DSS queries, the focus is to minimize the requirement of system resources essential to execute the distributed DSS query. For stochastic approaches, it is very difficult to confine the values of different GA parameters. In *SGQO, NGQO, RSQO and ERSQO*, several experiments are accomplished by varying the different parameters of genetic approach such as size or population, number of generations, crossover rate, and mutation rate. Empirically, it is observed that the optimal value of *Total Costs* for the above set of distributed database queries is obtained with the following statistics of genetic parameters [20,21].

Size of population (PopSize)	50
Number of generation (MaxGenr)	50
Crossover probability	0.3
Mutation probability	0.02

The results and analysis of DSS queries are described in the next section.

7. Results and discussions

An effort has been made to analyze and improve the design of stochastic DSS query optimizer [32]. A number of experiments are performed to optimize a set of distributed DSS queries by using exhaustive enumeration and four different stochastic approaches. The queries are optimized to reduce the usage of system resources requisite to execute it. In general, for a distributed DSS query, there are three types of system resources viz. input–output, processing and communication. *Total Costs* also known as 'Total Time' represents the combined usage of system resources required to execute the query [13]. Here, the focus is to improve the throughput of the stochastic query optimizer. The major objectives of this research work revolve around the following points:

- Analysis of different DSS query optimizer's viz. *EAQO, SGQO, NGQO, RSQO* and *ERSQO*.
- To examine the effect of data replication factor on the DSS query optimization process.
- To statistically analyze the relationship between number of join operations and the usage of system resources required to execute the distributed DSS query.

The remaining subsections explain the results of different distributed DSS query optimizers in context to the above said objectives.

7.1. Comparative analysis of different DSS query optimizers

An extensive experimentation was carried out on a set of distributed *DSS* queries. All the experiments were well performed under simulated environment. Table 2 represents the Total Costs for a set of distributed DSS queries obtained using the above said query optimization approaches. The Total Costs is measured in seconds. To avoid any discrepancy in the results of stochastic approaches, an experiment is performed several times and the average of the results has been taken.

From Table 2, it is observed that in general, *Total Costs* of distributed DSS query increases when number of join operations is increased. It was observed that the *Total Costs* of distributed *DSS* query produced by *ERSQO* approach is consistently less than that of stochastic approaches (*SGQO, NGQO, RSQO*) and *Entropy based Restricted Stochastic Query Optimizer*. In other words, *EAQO* always found the optimal solution for *Operation Site Allocation* problem. Fig. 2 represents the values of Total Costs of a set of distributed DSS queries obtained when the queries are optimized using four different approaches viz. *SGQO, NGQO, RSQO* and *ERSQO*.

Fig. 2 represents the quality of solution in terms of *Total Costs* with respect to the results quality of exhaustive enumeration approach. From Fig. 3, observation is that the solutions of operation site allocation problem of *DSS* query obtained using *SGQO* are 20% less optimal than exhaustive enumeration query optimizer. *Novel Genetic Approach* of 'Sevinc and Cosar' improved the solution quality in terms of Total Costs of SGQO up to 5%. The optimal solution produced by *NGQO* is 15% less optimal than the solution produced by EAQO. *RSQO* also improved the results of *NGQO* by up to 3%, i.e. the solutions produced by *RSQO* are 12% less optimal than that of *EAQO*. *ERSQO* further improved the solution quality of *RSQO* by 5%. Thus, the quality of the results produced using *ERSQO* is very close to that of *EAQO*. To summarize, by using *Entropy based Restricted Stochastic Query Optimizer*, the *Total Costs* of *DSS* query in distributed database

Table 2 Analysis of total costs using different stochastic approaches.

S. no.	Query	Total costs using EAQO (s)	Total costs using SGQO (s)	Total cost using NGQO (s)	Total costs using RSQO (s)	Total costs using ERSQO (s)
1	DSS1	500,100	575,115	550,110	535,107	509,510
2	DSS2	1,167,690	1,342,844	1,284,459	1,249,428	1,190,765
3	DSS3	1,655,630	1,903,975	1,821,193	1,771,524	1,681,635
4	DSS4	2,127,680	2,468,109	2,340,448	2,276,618	2,172,080
5	DSS5	2,409,105	2,794,562	2,698,198	2,577,742	2,487,031
6	DSS6	2,883,885	3,374,145	3,258,790	3,114,596	3,024,110
7	DSS7	3,361,605	3,966,694	3,865,846	3,697,766	3529685.3
8	DSS8	3,885,690	4,585,114	4,468,544	4,274,259	4,109,310
9	DSS9	4,573,165	5,487,798	5,213,408	5,121,945	4847554.9
10	DSS10	5,119,432	6,143,318	5,887,347	5,733,764	5526597.9

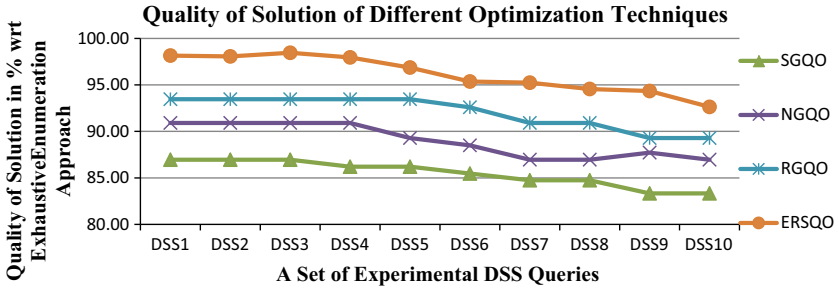

Figure 2 Stochastic versus exhaustive enumeration query optimizer.

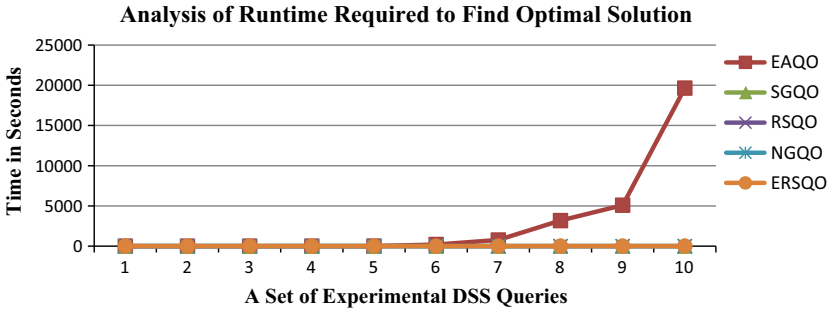

Figure 3 Runtime analysis of different stochastic approaches.

environment was reduced up to 13%, 8% and 6% as compared to Rho and March (*SGQO*), Sevinc and Cosar (*NGQO*) and *Restricted Stochastic Query Optimizer* (*RSQO*) respectively.

Fig. 3 represents the runtime required to find an optimal solution for operation site allocation problem of DSS query using different approaches.

From Fig. 3, it is concluded that Exhaustive Enumeration (EAQO) strategy can be used for simple DSS queries, as the runtime increased at an incredible rate for large and complex DSS queries. However, the runtime of SGQO, NGQO, RSQO and ERSQO remained constant or increased very slowly. Therefore, independent of the number of join operations, all the different stochastic genetic approaches work effectively for both simple and complex queries. Moreover, from Fig. 3, it is clear that the curve for all the stochastic query optimizer lies on a single line, as there is insignificant or no variation in their corresponding values.

7.2. Analysis of the effect of data replication over total costs

Initially, the data were supposed to be replicated on two different sites of a distributed database system. In this section, an effort has been made to examine the effect of data replication on the optimization process of the distributed decision support system queries. Here, two extreme cases of data replication are considered as given below:

Case I: When 20% data is replicated.
Case II: When 90% of data is replicated.

Several experiments are performed to examine the effect of data replication on the usage of system resources required to execute the query i.e. the *Total Costs* of distributed DSS query. The experiments are carried out by using four different

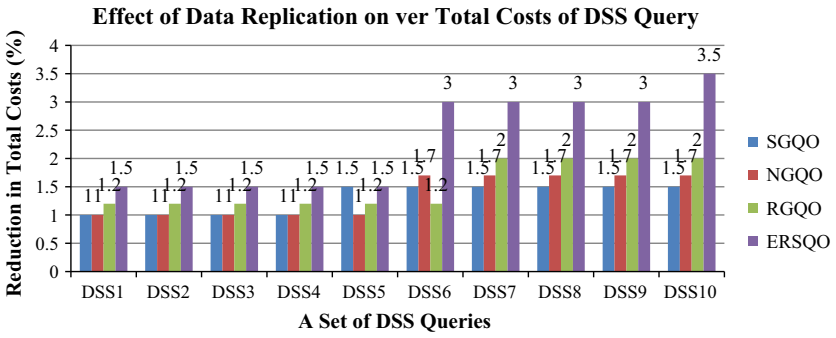

Figure 4 Analysis of effect of replication factor over total costs of DSS queries.

Table 3 Correlation between total costs and number of join operations.

S. no.	Number of join operations	Total costs using ERSQO
1	1	509,510
2	2	1,170,765
3	3	1,661,635
4	4	2,172,080
5	5	2,487,031
6	6	3,024,110
7	7	3,393,015
8	8	4,199,310
9	9	4,606,150
10	10	5,222,314

stochastic approaches viz. *SGQO, NGQO, RSQO* and *ERSQO*. From the experimental results, it was found that by increasing the replication rate from 20% to 90%, the *Total Costs* of a *DSS* query can be further optimized. For a set of experimental DSS queries, the *Total Costs* was reduced by 1.5%, 1.7%, 2% and 3.5% by using *SGQO, NGQO, RSQO* and *ERSQO* respectively. Fig. 3 represents the effect of high replication factor on different stochastic DSS query optimizers.

From Fig. 4, it is observed that entropy based Restricted Stochastic Query Optimizer gives better results, when the query was optimized with 90% replication factor.

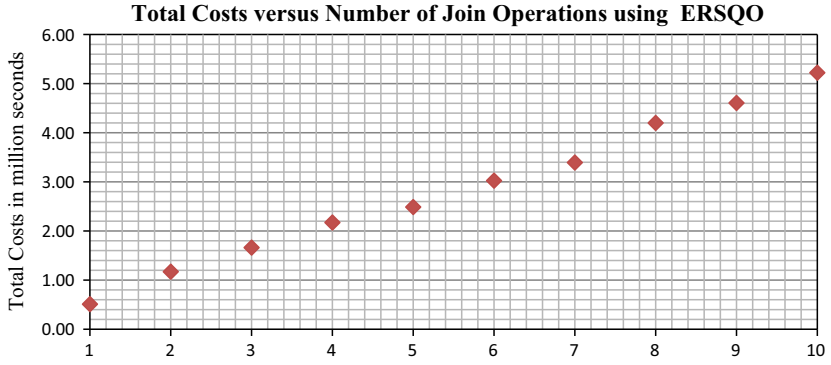

Figure 5 Scatter diagram of total costs and number of join operations.

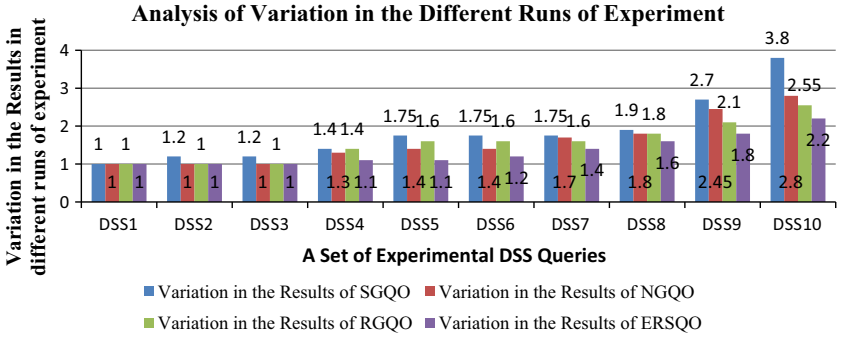

Figure 6 Variation analysis of different DSS query optimizers.

7.3. Analysis of correlation between number of join operations and total costs

Number of experiments has been conducted on a set of distributed *DSS* queries to establish the relationship between join operation and the *Total Costs* of query using *ERSQO*. Table 3 shows the value of number of join operations and the *Total Costs* of the distributed *DSS* queries.

Fig. 5 represents the scatter diagram between the number of join operations and the *Total Costs* of the *DSS* query as generated by *ERSQO*.

From Fig. 5, a positive correlation is remarked between a number of join operations and the *Total Costs* of *DSS* query as generated using *ERGA*. In this case, Pearson's Coefficient of Correlation(r) is 0.997. Therefore, a strong relationship between join operations and the *Total Costs* has been witnessed. To analyze the accuracy level of different stochastic DSS query optimizers, the analysis of variation has been carried out. The experiment for each DSS query is repeated ten times for different approaches and the variation among the different results of each experiment is examined. The study reveals that *Entropy based Restricted Stochastic DSS* query optimizer again outperforms other stochastic approaches. For a set of ten different experiments distributed *DSS* queries, the variation in the results is up to 3.8%, 2.8%, 2.55% and 2.2% for *SGQO, NGQO, RSQO and ERSQO* respectively. The variation is represented in Fig. 6.

8. Conclusion

Decision support system queries process massive amount of data (in *GigaBytes, PetaBytes* or even more). For a distributed DSS query, system resources required to execute the query is of major concern. Therefore, *DSS* queries are optimized on the basis of *Total Costs (Sum of input output costs, processing costs and communication costs*). The proposed system (*RSQO & ERSQO*) designs a model to reinforce the performance of a distributed *DSS* query. The intention was to quickly generate the best possible *DSS* query operation allocation scheme. A simulator was developed to decipher the operation site allocation problem for a set of distributed *DSS* queries. A set of adhoc DSS queries is examined based upon *Total Costs* and runtime by using different query optimizers viz. *EAQO, SGQO, NGQO, RSQO and ERSQO*. It is observed that the quality of solution (*Total Costs*) in finding an optimal query execution plan was not so good in *SGQO* and *NGQO* as compared to exhaustive enumeration approach. The proposed *RSQO* improves the quality of solution of *SGQO* and *NGQO* by 7% and 3% respectively. To further improve the quality of stochastic query optimizer, the idea of 'Havrda and Charvat' entropy has been incorporated with the stochastic approach. It was experimentally found that the use of Havrda and Charvat entropy improves the quality of *SGQO, NGQO and RSQO by 12%, 8% and 5%* respectively. Moreover, the *Total Costs* of the distributed *DSS* query is also affected by the replication factor of distributed database system. By increasing the replication rate from 20% to 90%, the *Total Costs* of *DSS* query was reduced by 1.5%, 1.7%, 2% and 3.5% by using *SGQO, NGQO, RSQO* and *ERSQO* respectively. It was ascertained that for distributed *DSS* queries, *Entropy based Restricted Stochastic Query Optimizer* better achieved the conflicting goals of high quality (*Total Costs*) and low time complexity (*Runtime*). In terms of solution quality, *ERSQO* outperformed *SGQO, NGQO and Restricted Stochastic Query optimizer* as well. In addition to this, a statistical analysis of number of join operations and the *Total Costs* of *DSS* query shows that a positive correlation exists between the number of join operations and the *Total Costs* of *DSS* query. Moreover, ERSQO shows more consistent results as compared to other stochastic query optimizers.

9. Future scope

Further, research is needed to be carried out to automate the transformation process in which a 'SQL' based query automatically generates its query tree before the optimization process starts. The design of entropy based stochastic *DSS* query optimizer can be further improved by examining the effect of different selection techniques of genetic approach. The results of the proposed approach can be compared and contrasted with other nature inspired evolutionary optimization techniques. One can also analyze the effect of data allocation and access policies for optimization operation of *DSS* queries.

Conflict of interest

Authors declare no conflict of interest.

References

[1] Hevener AR, Yao SB. Query processing in distributed database systems. IEEE Trans. Softw. Eng. 1979;5(3):177–87.
[2] Ceri S, Pelagatti G. Allocation of operations in distributed database access. IEEE Trans. Comp. 1982;31(2):119–29.
[3] Chen Yan, Zhou Lin, Li Taoying, Yu Yinging. The semi-join query optimization in distributed database system. In: National Conference on Information Technology and Computer Science. Atlantis Press; 2012. p. 606–9.
[4] Martin TP, Lam KH, Russel Judy I. An evaluation of site selection algorithm for distributed query processing. Comp. J. 1990;33(1):61–70.
[5] Apers Peter MG, Hevner Alan N, Yao Bing S. Optimization algorithms for distributed queries. IEEE Trans. Softw. Eng. 1983; SE-9.1:57–68.
[6] Ghaemi Reza, Fard Amin Milani, Tabatabaee Hamid, Sadeghizadeh Mahdi. Evolutionary query optimization for heterogeneous distributed database systems. World Acad. Sci., Eng. Technol. 2008;2:34–40.
[7] Mor Jyoti, Kashyap Indu, Rathy RK. Analysis of query optimization techniques in databases. Int. J. Comp. Appl. 2012;47(15):5–9.
[8] Sevinc Ender, Cosar Ahmat. An evolutionary genetic algorithm for optimization of distributed database queries. Comp. J. 2011;54 ():717–25.
[9] Kayvan Asghari, Ali Safari Mamaghani, Mohammad Reza Meybodi, An evolutionary algorithm for query optimization in database, in: Innovative Techniques in Instruction, E-Learning, E-Assessment and Education, 2008, pp. 249–254.
[10] Chande Swati V, Sinha Madhvi. Genetic algorithm: a versatile optimization tool. BVICAM's Int. J. Inf. Technol. 2008;1(1):7–12.
[11] Panicker Shina, Vijay Kumar TV. Distributed query plan generation using multi-objective genetic algorithms. World Scient. J. 2014;2014:1–17.
[12] Johann Christoph Fregtag, The Basic Principles of Query Optimization in Relational Database Management System, Euro-

pean Computer Industry Research Centre Germany, Internal Report IR-KB-59, 1989, pp. 1–15.
[13] M. Tamer Ozsu, Valduries Patrick, Principles of Distributed Database System, second ed., Pearson Education (chap. 1–6).
[14] March, Rho ST. Allocating data and operations to nodes in distributed database design. IEEE Trans. Knowl. Data Eng. 1995; 7(2):305–17.
[15] Kumar TV, Singh V, Verma AK. Distributed query processing plan generation using genetic algorithm. Int. J. Comp. Theory Eng. 2011;3(1):38–45.
[16] Goldberg David E. Genetic Algorithm in Search, Optimization & Learning. New Delhi: Pearson Education; 1999 (chap. 1).
[17] Paulinas Mantas, Ušinskas Andrius. A survey of genetic algorithms applications for image enhancement and segmentation. Inf. Technol. Control 2007;36(3):278–84.
[18] Carlos Alberto Gonzalez Pico, Roger L. Wainwright, Dynamic scheduling of computer tasks using genetic algorithms, in: Proceedings of the First IEEE Conference on Evolutionary Computation IEEE World Congress on Computational Intelligence, Orlando, 1994, pp. 829–833.
[19] Omara Fatma A, Arafa Mona M. Genetic algorithm for task scheduling problem. J. Paral. Distrib. Comput. 2010;70(1):13–22.
[20] Karegowda Asha Gowda, Manjunath AS, Jayaram MA. Application of genetic algorithm optimized neural network connection weights for medical diagnosis of Pima Indians diabetes. Int. J. Soft Comput. 2011;2(2):15–23.
[21] Hill Anthony M, Kang Sung-Mo. Genetic algorithm based design optimization of CMOS VLSI circuits. Lecture Notes in Computer Science 2005;866:545–55.
[22] Lienig J. A parallel genetic algorithm for performance-driven VLSI routing. IEEE Trans. Evolution. Comput. 1997;1(1):29–39.
[23] Man KF, Tang KS, Kwong S. Genetic algorithms: concept and applications. IEEE Trans. Indust. Electron. 1996;43(5):519–34.
[24] Du Jun, Alhajj Reda, Barker Ken. Genetic Algorithm based approach to database vertical partition. J. Intell. Inf. Syst. 2006;26:167–83.
[25] Kapoor JN. Measures of Information and Their Applications. Wiley Publishers; 1994.
[26] Zhou Rongxi, Cai Ru, Tong Guanqun. Applications of entropy in finance: a review. Entropy 2013;15(11):4909–31.
[27] Hien To, Kuorong Chiang, Cyrus Shahabi, Entropy-based histogram for selectivity estimation, in: CIKM, 2013, pp. 1939–1948.
[28] Cornell Douglas W, Yu Philip S. On optimal site assignment for relations in the distributed database environment. IEEE Trans. Softw. Eng. 1989;15(8):1004–9.
[29] Pramanik Sakti, Vineyard David. Optimizing join queries in distributed databases. IEEE Trans. Softw. Eng. 1988;14(9): 1319–26.
[30] TPS-DS Benchmark Report, 2012 <www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf> (accessed on 25/04/2013).
[31] Sarjo, Kapila, Kumar Dinesh, Kanika. A genetic algorithm with entropy based probabilistic initialization and memory for automated rule mining. Adv. Comp. Sci. Inf. Technol. Commun. Comp. Inf. Sci. 2011;131:604–13.
[32] Drenick PE, Smith EJ. Stochastic query optimization in distributed databases. ACM Trans. Database Syst. 1993;18(2): 262–88.